Aplicación que mejora el Patient Adherence

Barcelona, 18 de Octubre de 2016

Alumno: Alex Morral

Director: Xavi Guardia

Ponente: Carles Farrè

Facultad Informática de Barcelona

Sfy, Soft For You, S.L.

1. Resumen

Resumen

En el mundo médico, existe un problema con el que los profesionales del sector llevan lidiando desde hace muchos años, y todavía aún sigue siendo un reto a resolver. El *patient adherence* se define como el grado en que el comportamiento de la persona se corresponde con las recomendaciones acordadas por el profesional de la salud. La no adherencia desencadena un gran gasto innecesario en los recursos médicos y una peor calidad de vida de la persona, entre otras cosas.

La finalidad de este TFG es proporcionar una aplicación con la que las personas que padecen de enfermedades crónicas sientan un mayor interés a seguir con el tratamiento que se les ha asignado, y de esa manera mejorar su calidad de vida. La aplicación se centra en dos focos principales de personas: las que llevan tiempo con un tratamiento, y las que acaban de empezar con él.

La aplicación proporciona a las personas que llevan tiempo con el tratamiento un estímulo para seguir con el tratamiento mediante la posibilidad de ayudar a las personas que acaban de empezar un tratamiento. De esa manera, estas personas más experimentadas pueden ayudar a las menos experimentadas mediante consejos y experiencias que ellos mismos han vivido.

Por otra parte, la aplicación proporciona a las personas que acaban de empezar un tratamiento un lugar donde perder el miedo al nuevo tratamiento, ya que tienen la posibilidad de preguntar sus dudas y leer experiencias de personas con los mismos tratamientos. De esa manera se crea un vínculo entre las personas más experimentadas y las menos experimentadas ya que las dos pasan por una situación muy parecida.

Este proyecto se realiza en Sfy, empresa especializada en el desarrollo de soluciones tecnológicas para empresas, con Xavier Guardia como director, y con Carles Farrè como ponente de la Facultad de Informática de Barcelona

Resum

En el mon mèdic, existeix un problema amb el que els profesionals del sector porten lluitant des de fa molts anys, i encara ara continua essent un repte a resoldre. El *patient adherence* es defineix com el grau en que el comportament de la persona es correspon amb les recomanacions acordades pel profesional de la salut. La no adherencia desencadena un gran cost innecesari en els recursos mèdics i una pitjor qualitat de vida de la persona, entre altre coses.

La finalitat d'aquest TFG es proporcionar una aplicació amb el que les persones que pateixen d'enfermetats cròniques sentin un major interès en seguir amb el tractament que se'ls hi ha encomanat, i d'aquesta manera millorar la seva qualitat de vida. La aplicació se centra en dos focus principals de persones: les que porten temps amb un tractament, i les que acaben de començar amb ell.

La aplicació proporciona a les persones que porten temps amb el tractament un estímul per a seguir amb el tractament mitjançant la possibilitat d'ajudar a les persones que acaben de començar amb el tractament. D'aquesta manera, aquestes persones més experimentades poden ajudar a les menys experimentades mitjançant consells i experiencies que ells mateixos han viscut.

Per altra banda, la aplicació proporciona a les persones que acaben de començar un tractament un lloc on perdre la por a un nou tractament, ja que tenen la possibilitat de preguntar els seus dubtes i llegir experiencies de persones amb els mateixos tractaments. D'aquesta manera es crea un vincle entre les persones més experimentades i les menys experimentades ja que les dues pasen per una situació molt semblant.

Aquest projecte es desenvolupa a Sfy, empresa especialitzada en el desenvolupament de solucions tecnològiques per empreses, amb Xavier Guardia com a director, i amb Carles Farrè com a ponent de la Facultat d'Informàtica de Barcelona.

Abstract

In the medical world, there is a problem with which health professionals have been struggling for many years, and yet still remains a challenge to solve. The patient adherence is defined as the degree to which the behavior of the person corresponds to the recommendations agreed by the health professional. Nonadherence triggers a huge overspending on medical resources and a poorer quality of life of the person, among other things.

The purpose of this TFG is to provide an application to make people suffering from chronic illnesses feel greater interest to continue with the treatment that has been assigned to them, and improve their quality of life. The application focuses on two main types of people: those who have spent more time with the treatment and those that are just starting it.

The application provides people who have spent more time with the treatment with a stimulus to continue the treatment by the possibility of helping people who have just started the treatment. Thus, these more experienced people can help the less experienced through tips and experiences that they have lived.

On the other hand, the application provides people who have just started treatment a place where they can lose their fear to a new treatment because they have the possibility to ask their questions and read experiences of people with the same treatments. Thus a link between the most experienced and less experienced people is created since the two go through a very similar situation.

This project is developed at Sfy, company specialized in development of technological solutions for companies with Xavier Guardia as director, and Carles Farrè as speaker from the Facultat Informática de Barcelona

Índice

1. Contexto

1.1. Formulación del problema

El *patient adherence* se define como el grado en que el comportamiento de la persona se corresponde con las recomendaciones acordadas por el profesional de la salud.

La no adherencia ocurre cuando los pacientes deciden parar de seguir el tratamiento después de empezarlo, sin haber sido recomendado por el profesional de la salud.
La adherencia es el factor clave asociado con la efectividad de todas las terapias farmacológicas pero es particularmente crítico para medicaciones preescritas para condiciones crónicas.

La consecuencia de la no adherencia es un malgasto de medicación, progresión de la enfermedad, habilidades funcionales reducidas, una peor calidad de vida, un uso incrementado del uso de recursos medicos como podrian ser visitas al hospital y ingresos al hospital[1].

En los países desarrollados, la adherencia a las terapias a largo plazo en la población general está en un 50% y es mucho más baja en los países en vias de desarrollo[2].

"Drugs don't work in patients who don't take them"
C. Everett Koop

"Keep a watch also upon the faults of the patients, which also make them lie about the taking of things prescribed"
Hippocrates

La poca adherencia suele darse por una gran cantidad de causas. Seguidamente se enumeran algunas de ellas:

- *Olvido*. El paciente se olvida de tomarse la medicación.
- *Falta de conocimiento* de la enfermedad.
- *Falta de contribución* del paciente en el proceso de toma de decisiones del tratamiento.
- *Falta de motivación* del paciente por seguir con el tratamiento.
- *Falta de soporte* hacia el paciente por parte de familia y/o amigos.
- *Creencias* del paciente acerca de la efectividad del tratamiento.
- *Poco entendimiento* de las instrucciones de la medicación.
- *Miedo* a efectos secundarios.

El objetivo del proyecto es mejorar la adherencia de los pacientes hacia el tratamiento que les esté establecido. Sin embargo, debido a ser tener unas causas tan diversas y ser un problema tan personal para cada paciente, este trabajo se ha centrado en la falta de soporte y motivación que sienten los pacientes con enfermedades crónicas, que les impide continuar con su tratamiento, y que deriva así en una poca adherencia al tratamiento que les está establecido.

1.2. Objetivos

El objetivo de este TFG es el desarrollo de un sistema en el que las personas con tratamientos para enfermedades crónicas, puedan, por una parte introducir sus experiencias y consejos para que otras personas puedan leerlas, y por otra, que las personas puedan informarse, leer y debatir preguntas/experiencias/consejos. El sistema estará compuesto por:

Aplicación iOS
La aplicación que usarán las personas. Mediante esta aplicación, las personas podrán interactuar con los mensajes, de tal manera que podrán crear hilos, responderlos y a su vez contactar con otros usuarios mediante un chat.

API Backend
Se debe crear una API que se encargará de guardar toda la información del sistema. El backend y la API nos van a permitir que todos los dispositivos estén actualizados con la última información disponible.

Otro objetivo principal del proyecto, es el ejercer todos los roles del proyecto. Con todos los roles, me refiero tanto a Gestor del Proyecto, desarrollador de la aplicación y del backend, como diseñador de la interficie. De esa manera poder aplicar los conocimientos adquiridos en las asignaturas del grado, sobretodo de la especialidad de Desarrollo de Software.

1.3. Stakeholders

En este apartado se definen los principales actores de este proyecto, es decir, las partes interesadas que tienen un objetivo para con el proyecto. Es de gran importancia definirlos ya que cada uno tiene objetivos diferentes.

SFY
La empresa desarrolladora del proyecto, juntamente con el director de proyecto tienen un claro interés en que éste se realice. El director de proyecto se ha encargado de guiar y supervisar el trabajo del autor del proyecto, además de proporcionar los recursos necesarios para llevarlo a cabo.

Pacientes
Son los principales beneficiarios y los que van a usar el proyecto. A su vez, también son los actores a los que va dirigido este proyecto, y los que van a usarlo. Al cumplirse los objetivos de este proyecto, los pacientes pueden gozar de una mejora en su calidad de vida.

Profesionales de la salud
Los profesionales de la salud (médicos, doctores, etc) que tratan a los pacientes también son partes interesadas de este proyecto. Al cumplirse los objetivos del proyecto, podran asumir que sus pacientes tienen una ayuda para seguir su tratamiento y de alguna manera no estar tan encima de ellos.

1.4. Alcance

El sistema resultante de este proyecto se lleva a cabo a partir de la integración de dos componentes claramente identificados:

Aplicación iOS
La aplicación es el principal foco de lógica del proyecto. Es la encargada de gestionar la información del paciente y también será el foco de soporte y motivación para el paciente de tal manera que dispone de funcionalidades con el objetivo de aumentar esos aspectos en el paciente. Estas funcionalidades están pensadas para que los pacientes compartan sus experiencias y consejos con los demás pacientes que también disponen de la aplicación.

Así pues, podemos definir las funcionalidades de la aplicación:

- El paciente introduce la información referente a el mismo, pudiendo posteriormente editarla. Alguna de esta información es pública y podrá ser vista por otros usuarios.
- El paciente puede crear hilos en los que introducir experiencias y/o consejos sobre tratamientos o medicación. Esta información puede ser borrada por el paciente.
- El paciente pueden conectarse con otros pacientes de manera que los pacientes puedan hablar mediante un chat entre ellos.

Backend Laravel

El backoffice es el encargado de gestionar todas las peticiones que se realicen a través de la aplicación. Básicamente consta de un servidor compuesto por una API y una base de datos. La aplicación se comunica a través de la API para recibir la información de la base de datos del servidor y así disponer de la información más actualizada.

2. Estado del arte

2.1. Sistemas actuales

AdhereTech[3]

Cómo dicen en su página web, “Adheretech es una firma de Salud que crea herramientas simples y afectivas para el *adherence”*.

El producto de Adheretech es una caja de pastillas wireless y inteligente que recoge, envía y analiza datos en tiempo real. De esta manera, si las dosis preestablecidas por un paciente no han sido tomadas, estos pueden recibir alertas y intervenciones (llamadas de teléfono, SMS, etc).

Imagen Adheretech

Medisafe[4]

Medisafe es una aplicación para iOS y Android cuyo objetivo principal es recordar al paciente la medicación que debe tomar. Además, también permite mantener información sobre la medicación que está tomando el paciente. Un punto importante de esta herramienta, es que cuenta con un apartado llamado Medfriend cuyo objetivo es conectar a los pacientes con profesionales de la salud, familiares o amigos de tal manera que estén enterados de las dosis que no se toman.

Imagen Medisafe

Medhelper App[5]

Medhelper App es una aplicación para iOS y Android cuyo objetivo es el de organizar las tomas de dosis de medicación facilmente. Además, tambien consta de un sistema de alertas para avisar de las tomas así cómo un historial de medicaciones que no han sido tomadas.

e-Pill Medication Reminders[6]

Empresa que se dedica a fabricar varios dispositivos relacionados con el *adherence*. Basicamente, sus dispositivos se pueden agrupar en dos grupos: Los que te ayudan a recordar las tomas de medicación y los que te ayudan a organizar las medicaciones, ya sea mediante compartimentos para cada día, toma, hora, etc. o automáticamente.

DoseCast[7]

DoseCast es una aplicación para iOS y Android que te permite introducir las tomas de medicamentos para posteriormente dar de alta alertas que te sirvan para recordar que tienes que tomar la medicación. Además la aplicación guarda información sobre las tomas de medicación y alertas cuando hay que rellenar alguna medicación.

2.2. Conclusiones

Cómo hemos visto, generalmente todos los productos en el mercado, se centran en una de las causas por las que se da el poco *adherence* de los pacientes, el *olvido*. Los productos intentan recordar al paciente que debe tomarse la dosis y en algúnos casos, qué dosis debe tomarse. Podriamos decir que la funcionalidad principal de todas las aplicaciones es la de crear alertas con las que recordar las tomas de medicamentos.

En mi opinión, cabe destacar que Adheretech y Medisafe son los productos finales mejor implementados, pero tienen ciertas peculiaridades que se deben tener en cuenta, y que pueden aplicarse a otros productos anteriormente mencionados:

- Adheretech sólo está centrada exclusivamente en la causa del *olvido,* es decir, no se centra en otras causas de poco *adherence.* Técnicamente una botellita sólo te sirve para un tipo de pastillas. De esta manera, si el paciente debe tomarse más de un medicamento, ya sean otro tipo de pastillas o otro tipo de medicamento, no es tan ideal.

- Considero a Medisafe, el enfoque más parecido a este proyecto, ya que no sólo se centra en la casua del *olvido*, sino que además también se centra en el soporte social hacia el paciente. Aún así, mi opinión es que el modo en que se obtiene este soporte no es el adecuado.

La diferencia entre las soluciones existentes y la que se propone es que generalmente están centrados en recordar al paciente la toma de medicamentos y en algunos casos, específicamente en pastillas. A excepción de Medisafe, en las soluciones existentes no se tienen en cuenta ninguna de las demás causas del poco adherence de los pacientes.

El proyecto que se propone, no se centra en recordar la toma de medicación de cualquier tipo, sino que dar soporte y motivación al paciente son objetivos principales del producto. Mediante la creación de una comunidad de personas con unas experiencias parecidas, se llega al usuario mucho más, ya que los consejos y experiencias que puedan recibir de personas que han pasado por circunstancias similares serán de mucho más valor.

3. Analisis de requisitos

En este apartado se definiran los requisitos funcionales y no funcionales del sistema. Para una primera versión de la aplicación solamente va a existir un rol: usuario. Esta versión no va a contar con otros roles como podrían ser administrador o gestor de contenido. Esto es debido a la carga de trabajo que lleva el proyecto en sí. Si posteriormente en otras versiones de la aplicación se considerase añadir otros roles de usuario al sistema, se podrían añadir más requisitos funcionales.

3.1. Requisitos funcionales

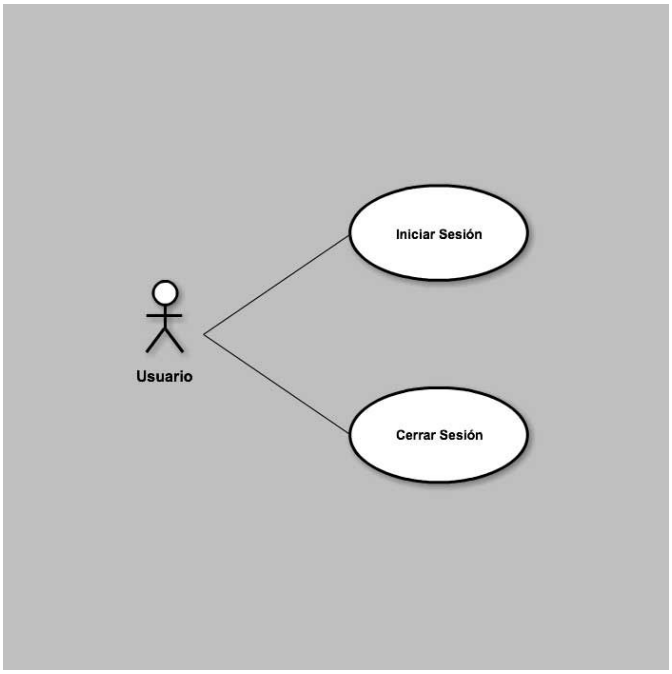


Imagen Casos de uso 1

3.1.1. Iniciar sesión

Actor: Usuario
Precondición: El usuario existe en el sistema
Descripción: El usuario podrá acceder a la aplicación mediante sus credenciales. Una vez autentificado, la aplicación mostrará la página principal en la que se muestran todos los Posts.
Casos de error: Los datos introducidos para el inicio de sesión no son correctos.

3.1.2. Cerrar sesión

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación
Descripción: El usuario podrá finalizar su sesión en la aplicación. La aplicación mostrará la página de inicio de sesión.
Casos de error: -

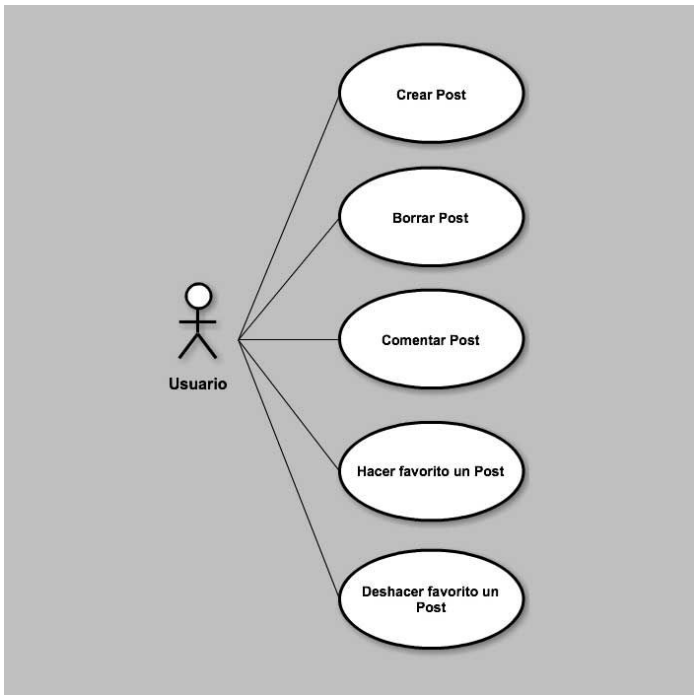


Imagen Casos de uso 2

3.1.3. Crear un Post

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación
Descripción: El usuario podrá crear un nuevo post. Deberá llenar la información correspondiente: título, descripción y tipo de post. Al crearlo, el post será creado en el sistema para poder ser visto por todos los usuarios.
Casos de error: La información introducida no es correcta o es vacía.

3.1.4. Borrar un Post

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación. El post debe pertenecer al usuario.
Descripción: El usuario podrá borrar un post que él mismo haya creado. El sistema pedirá confirmación para borrar ese post. Al aceptar la confirmación, el post será borrado del sistema.
Casos de error: -

3.1.5. Comentar un Post

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación. Debe existir un post.
Descripción: El usuario podrá comentar en cualquier post. Deberá introducir el texto que desee responder a un post. El sistema pedirá confirmación para comentar en el post. Al aceptar la confirmación, el comentario será realizado en el post.
Casos de error: La información introducida no es correcta o vacía.

3.1.6. Hacer favorito un Post

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación. El post no debe ser favorito.
Descripción: El usuario podrá asignar un post como favorito, para posteriormente verlo en otra sección de la aplicación.
Casos de error: -

3.1.7. Deshacer favorito un post

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación. El post debe ser favorito.
Descripción: El usuario podrá desasignar un post como favorito. El post, se eliminará de los favoritos del usuario.
Casos de error: -

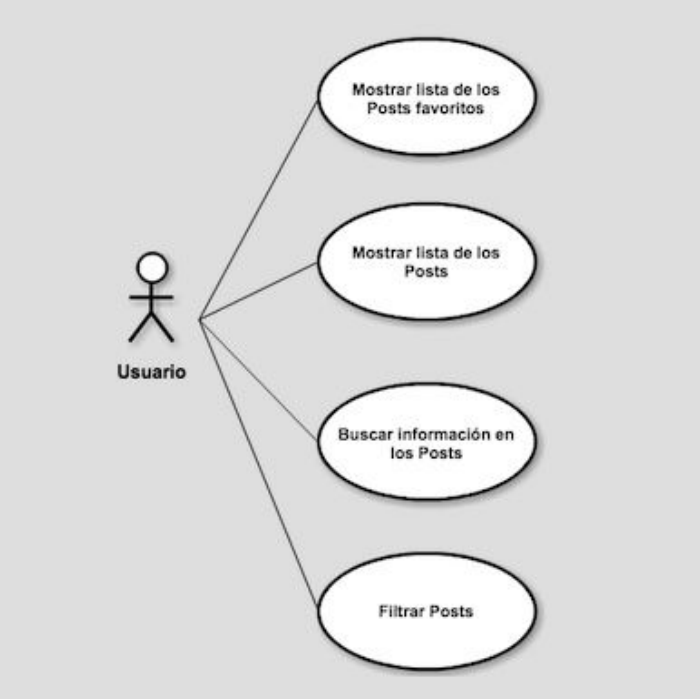


Imagen Casos de uso 3

3.1.8. Mostrar la lista de los posts

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación
Descripción: El usuario podrá consultar la lista de todos los posts. Además también podrá seleccionar cada uno de ellos para ver su información.
Casos de error: -

3.1.9. Mostrar la lista de los posts favoritos

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación
Descripción: El usuario podrá consultar la lista de todos sus posts favoritos. Además también podrá seleccionar cada uno de ellos para ver su información.
Casos de error: -

3.1.10. Mostrar los comentarios para un post

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación. Existir un post
Descripción: El usuario podrá consultar el detalle del post, en el que se mostrará la lista de comentarios de ese post. Desde esa pantalla podrá ir a la pantalla para realizar un comentario.
Casos de error: -

3.1.11. Buscar información en los Posts

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación
Descripción: Mediante un campo de búsqueda, el usuario podrá buscar información en los posts. Las palabras que introduzca en la búsqueda, se filtrarán en toda la lista de posts para ver si alguna información concide. Mostrará la lista de los posts que hayan coincidido con la búsqueda.
Casos de error: -

3.1.12. Filtrar Posts

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación
Descripción: Mediante una pantalla, el usuario seleccionará que categorías y tipos quiere mostrar en la lista. El usuario podrá seleccionar o deseleccionar los ítems que aparezcan. Mostrará la lista de los posts que hayan coincidido con el filtro.
Casos de error: -

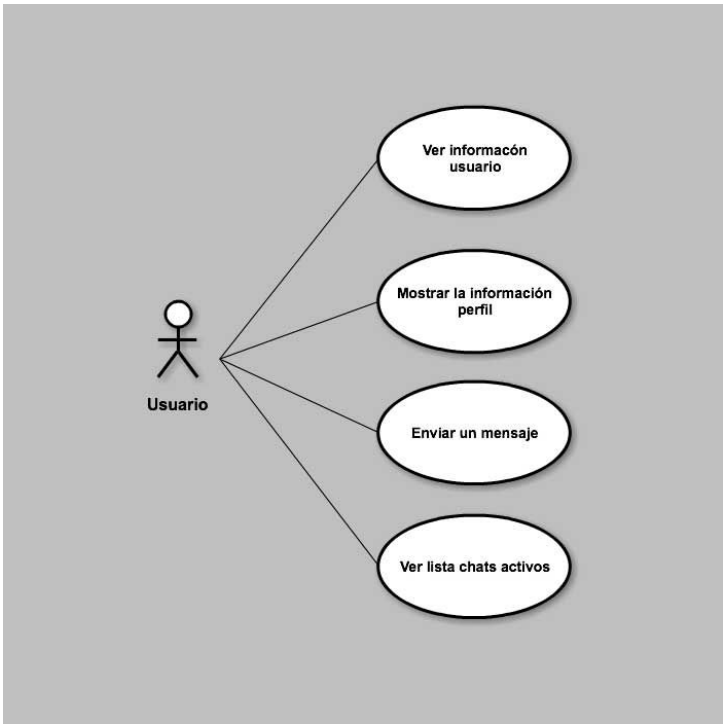


Imagen Casos de uso 4

3.1.13. Ver información de otros usuarios

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación
Descripción: El usuario podrá consultar la información de los demás usuarios. Al usuario se le mostrará el perfil con la información del usuario que seleccione.
Casos de error: -

3.1.14. Mostrar la información del perfil

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación
Descripción: El usuario podrá consultar la información de su propio perfil
Casos de error: -

3.1.15. Enviar un mensaje a otro usuario

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación. Existir un usuario al que enviarle un mensaje
Descripción: El usuario podrá enviar mensajes desde la pantalla de chat. Deberá introducir un texto para enviar y presionar el botón de enviar.
Casos de error: La información introducida no es correcta o es vacía.

3.1.16. Ver la lista de chats activos

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación.
Descripción: El usuario podrá enviar consultar la lista de chats que tiene activos con otros usuarios.
Casos de error: -

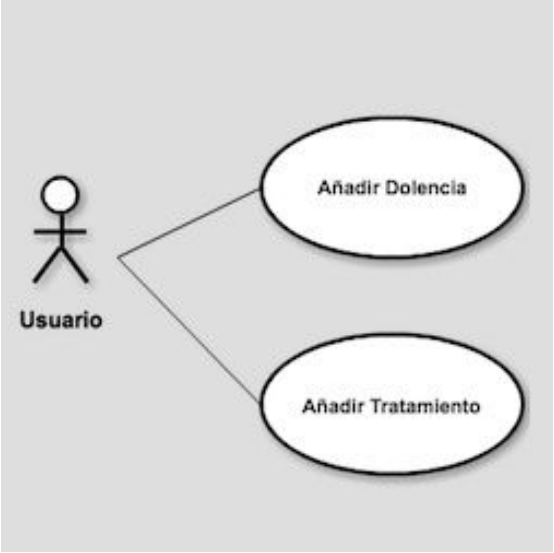


Imagen Casos de uso 5

3.1.17. Añadir Dolencia

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación.
Descripción: El usuario podrá añadir una dolencia a su perfil de usuario de manera que quede grabado en los datos del usuario.
Casos de error: -

3.1.18. Añadir Tratamiento

Actor: Usuario
Precondición: Haber iniciado sesión en la aplicación.
Descripción: El usuario podrá añadir un tratamiento a su perfil de usuario de manera que quede grabado en los datos del usuario.
Casos de error: -

3.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que permiten un buen funcionamiento del software. Estos tienen en cuenta requisitos de calidad como la eficiencia, el rendimiento, la seguridad y la fiabilidad del software

Usabilidad: La usabilidad en un sistema de software siempre es un tema importante. Se ha ajustado la aplicación a un diseño no muy creativo y siguiendo en gran medida los iOS Human Interface Guidelines[8] propuestos por Apple. De esa manera, se mantiene en gran medida la usabilidad ya que los usuarios no deben pensar la manera en que se usa la aplicacion, ya que se usa de la misma forma que las demás aplicación es de sistema.

Escalabilidad: El sistema se ha diseñado de tal manera que el gran volumen de información lo mantiene el servidor. De esa manera si llegara el caso en que se generara un volumen muy grande de información, no debería realizarse ningún cambio en la aplicación, solamente desplegar el backend en otro servidor.

Bajo tiempo de respuesta: La aplicación realiza un caché de las peticiones que realiza. De esa manera se garantiza que no se deba procesar mucho volumen de datos si estos ya se encuentran en el dispositivo. Solamente se deberán procesar los nuevos datos.

3.3. Modelo conceptual de datos

En este apartado se describe el modelo de datos diseñado mediante un diagrama.

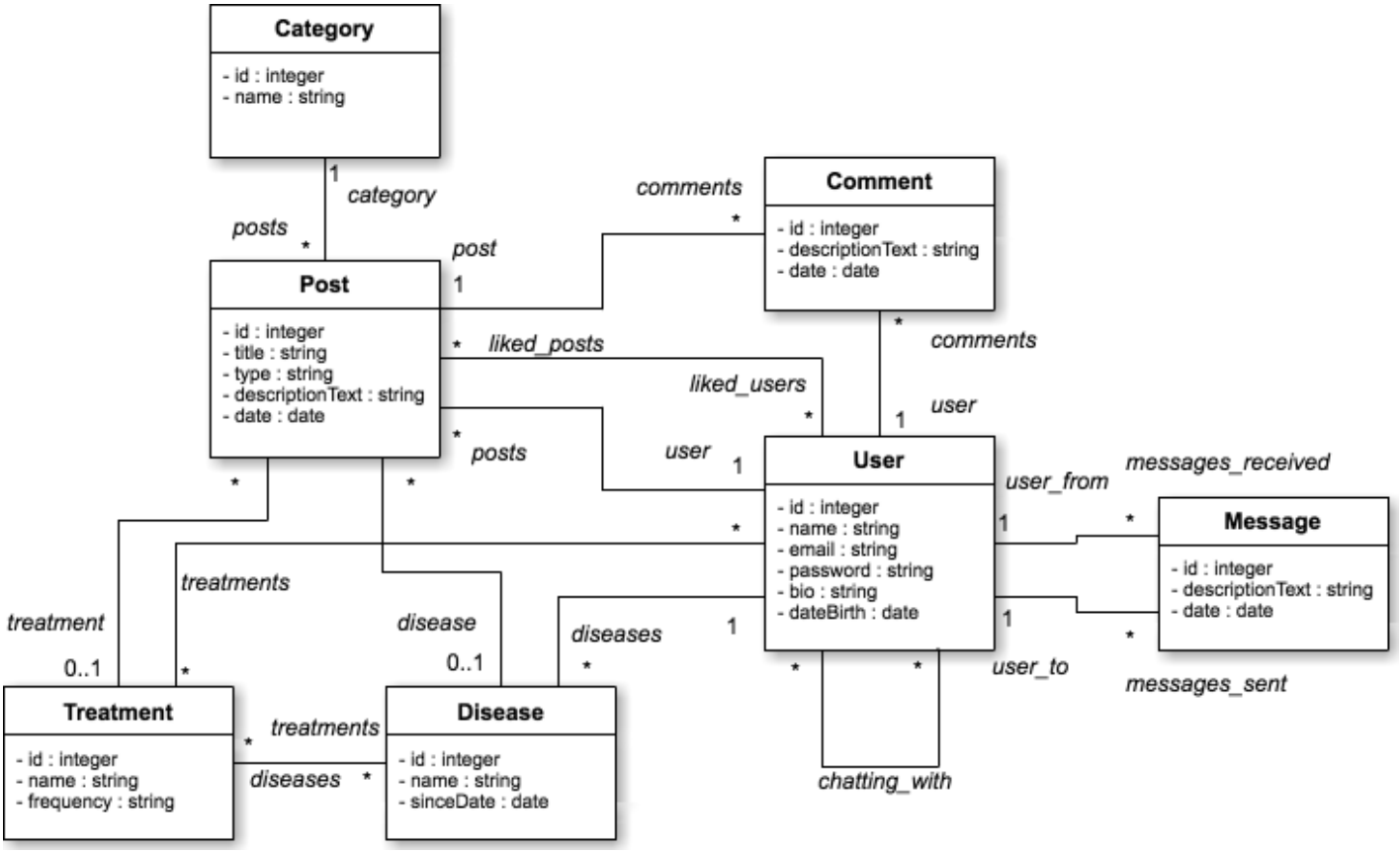


Imagen Modelo conceptual de datos

3.3.1. Descripción textual del modelo conceptual de datos

Los usuarios (User) son el eje central del modelo y se componen por el nombre, la contraseña, un email, una biografía y la fecha de nacimiento. Los usuarios están relacionados con los demás modelos de datos.

Por una parte se relacionan con los posts (Post). Los posts se componen por un título, una descripción, un tipo y una fecha. El modo en que se relacionan con los usuarios es *posts* que nos indica los posts que ha realizado ese usuario. El segundo modo son los *liked_posts* que nos indica qué *posts* son los favoritos de un usuario. Estas dos relaciones tienen relaciones inversas. La relación *posts* tiene como relación inversa *user*, que determinará el usuario que ha creado el post. Por otra parte, la relación *liked_posts* tiene como relación inversa *liked_users*, que nos va a indicar en cada post, los usuarios a los que les gusta ese post. A su vez, este modelo dispone de una relación *category* con el modelo *Category* que dirá a qué categoría pertenece ese post. Esta relación tiene cómo inversa la relación *posts* en *Category*, que nos indicará por cada categoría, los posts que contiene. Y por último tenemos las relaciones *disease* y *treatment* que pueden ser nulas. De esa manera relacionamos un *Post* con el *Treatment* o *Disease* del que se habla.

Los usuarios, por otra parte, también se relacionan con los comentarios (Comment). Los comentarios se componen por los campos fecha y descripción. Podemos obtener los comentarios que ha hecho cada usuario mediante la relación *comments.* Esta relación tiene como relación inversa la relación *user*, la cual nos dará el usuario que ha escrito un comentario.

Además nos encontramos que los usuarios también se relacionan con los mensajes (Message). Los mensajes se componen por los campos fecha y descripción. Las relaciones *messages_received* y *messages_sent* nos van a indicar los mensajes que se han recibido y enviado respectivamente por cada usuario. Estas dos relaciones tienen sus respectivas relaciones inversas, y por cada mensaje, podremos saber de qué usuario viene mediante la relación *user_from* y a qué usuario se envía mediante la relación *user_to.*

Por último tenemos los modelos Category, Disease y Treatment que cómo hemos explicado anteriormente se relacionan con Post y User y nos indicarán los detalles del Post y del Usuario respectivamente.

La última relación que nos queda es la de *comments* y *post*, de Post y Comment respectivamente. Estas relaciones son inversas entre ellas y nos van a indicar, por una parte los comentarios de un Post, y por la otra, el post padre de un comentario.

4. Diseño

4.1. Arquitectura

La arquitectura de la aplicación sigue el patrón MVC (*Model View Controller*). Este patrón consite en separar la lógica de negocio de la presentación de la información mediante tres capas con responsabilidades diferentes.

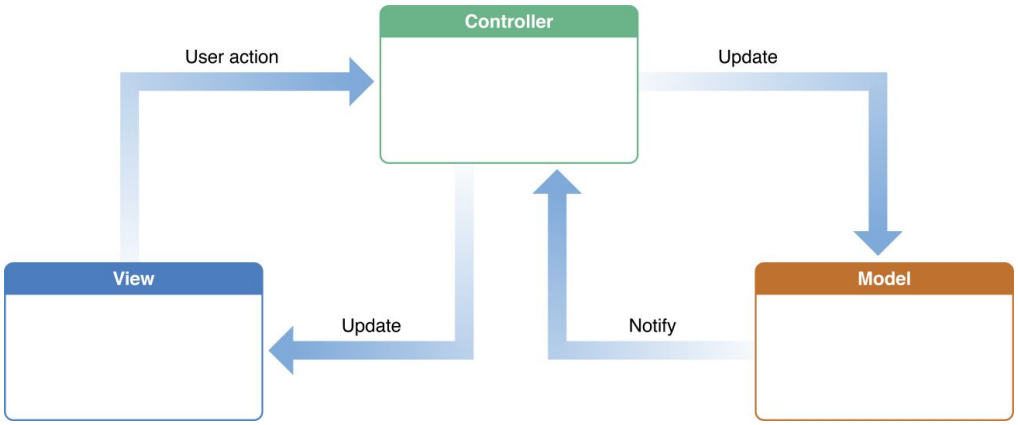


Imagen MVC [9]

Modelo

Es la capa donde se definen las entidades de información que estamos reflejando del mundo real. Esta capa se encuentra justo por encima de la base de datos y por lo tanto realiza un mapeo entre esta y los objetos del mundo real. En el caso de la aplicación, hemos usado CoreData, que es una framework proporcionado por Apple que nos proporciona base de datos y mapeo directo a las entidades del modelo.

Vista

Es la capa más cercana al usuario final ya que es la encargada de mostrar la información del sistema al usuario o agentes externos. En el caso de la aplicación, hemos usado *Interface Builder* para generar esta capa, ya que esta herramienta integrada en Xcode, nos permite realizar el diseño de las pantallas de la aplicación.

Controlador

Es la capa encargada de recibir y procesar las peticiones recibidas por parte del usuario o agentes externos (la API por ejemplo). El controlador recibe la petición externa y envia comandos al modelo para actualizar su estado. Una vez modificado su el estado, envia los comandos a la capa de la vista para actualizar la presentación de la pantalla.

En el caso de la API backend, Laravel se acostumbra a desarrollar con la misma arquitectura MVC. Sin embargo, al no disponer de componentes visuales, solamente dispondremos del Modelo y el Controlador.

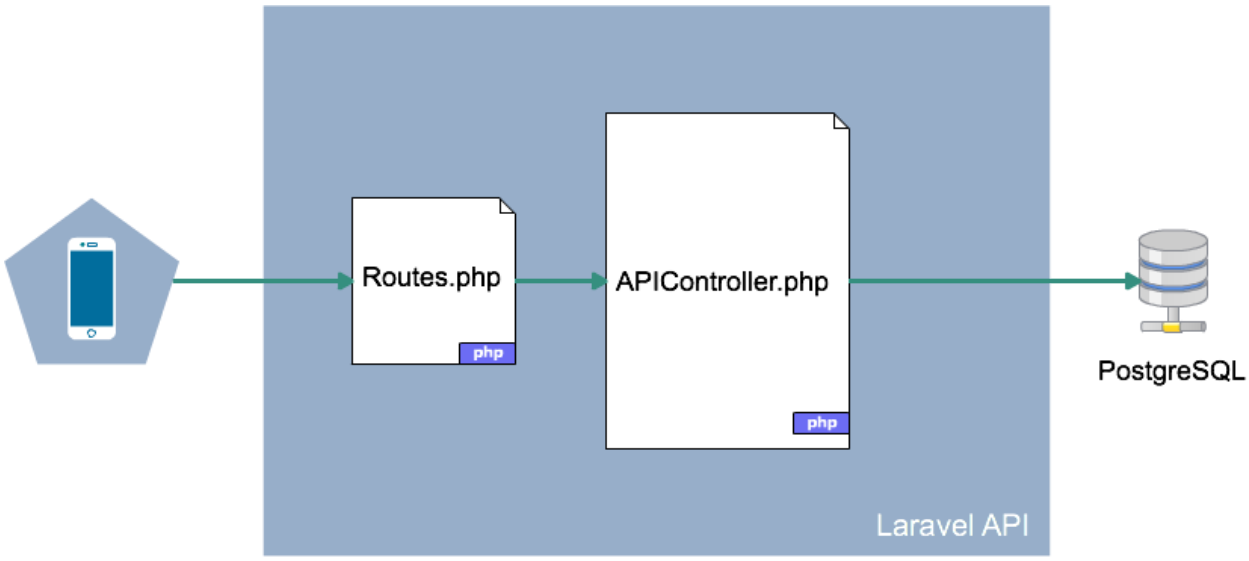
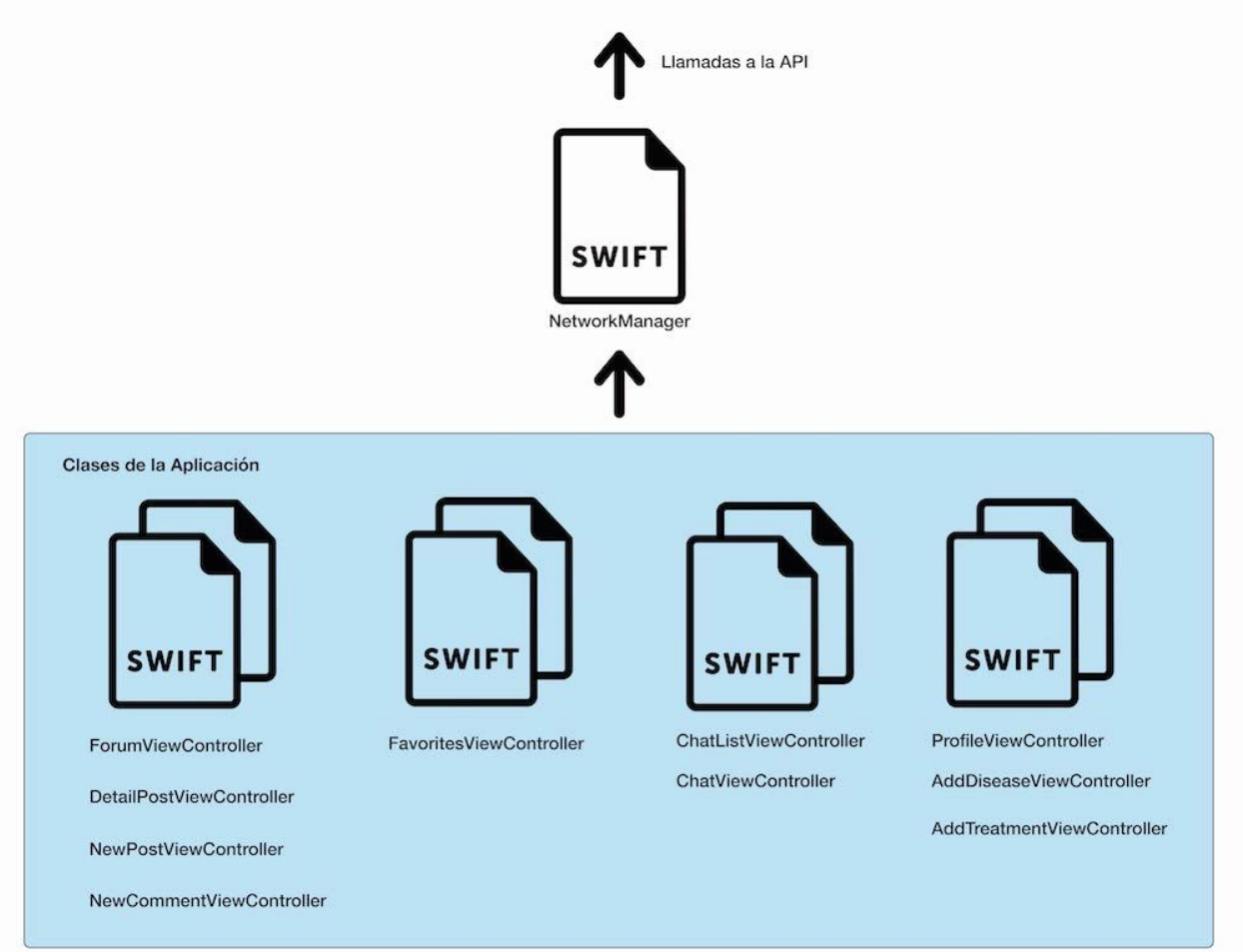


Imagen Arquitectura

4.2. Arquitectura Aplicación

A continuación mostramos un esquema del diseño interno de la aplicación:



En este esquema podemos ver las clases principales de la aplicacion que se encargan de realizar la logica y mostrar la vista correspondiente. Además de estas clases, hay algunas más menos importantes, las cuales se encargan de la lógica de navegación, celdas de las tablas, y clases predeterminadas de iOS.

Como apreciamos en el esquema, todas las clases, hablan con la clase *NetworkManager*, que es la encargada de realizar las llamadas a la API. Así pues, la clase *NetworkManager* es de tipo Singleton para mantener la coeherencia en toda la aplicación.

En la parte inferior del esquema vemos cómo las clases están divididas en columnas. Estas columnas representan las partes de la aplicación correspondientes a la *TabBar* inferior.

A continuación detallamos los métodos más importantes de las clases.

4.2.1. ForumViewController

ViewController que se encarga de mostrar la lista de posts de la aplicación. Es la primera pestaña de la TabBar inferior. Desde ella se accede al detalle de un post. Se puede marcar un post como favorito o eliminarlo. Desde aquí tambien se accede al filtro de posts y a la ventana de añadir un post.

func getDataFromBackoffice()

Este método es el que se encarga de obtener los datos del backend. Este método llama al singleton NetworkManager que posteriormente hace la petición a la API. Al acabar actualiza la tabla de posts.

func swipeTableCell(cell: MGSwipeTableCell!,
tappedButtonAtIndex index: Int,
direction: MGSwipeDirection, fromExpansion: Bool) -> Bool

Este método es un callback cuando se pulsa un botón de los que aparecen al realizar un movimiento Swipe sobre alguna celda. Marca el post como favorito/no favorito o muestra la alerta de borrado dependiendo del botón que se pulse.

func searchText()

Este método se llama al realizar una búsqueda en la barra de búsqueda. El método busca el texto en alguno de los campos de los post: title, descriptionText o user.name.

4.2.2. DetailPostViewController

ViewController encargado de mostrar el detalle de un post así como los comentarios. Desde este ViewController se accede al ViewController que permite añadir un comentario.

func getDataFromBackoffice()

Este método es el que se encarga de obtener los datos del backend. Este método llama al singleton NetworkManager que posteriormente hace la petición a la API. Al acabar actualiza la tabla de posts.

func goNewCommentVC()

Este método se encarga de ir al ViewController correspondiente a la creación de un nuevo comentario. En este caso NewCommentViewController.

func authorButtonPressed(sender: AnyObject?)

Este método se llama al pulsar sobre el autor del post. Al llamarse, lleva al usuario a la vista de perfil mostrando el perfil del autor del post. En este caso ProfileViewController.

4.2.3. NewPostViewController

ViewController encargado de realizar un nuevo post. Al rellenar los datos de la pantalla se realiza la petición a la API para crear un nuevo post en la aplicación.

func sendPostToBackoffice()

Es el método que se llama al aceptar el envío de un nuevo post. Llama al método correspondiente del singleton NetworkManager. Al recibir la respuesta, vuelve a la pantalla anterior. En este caso ForumViewController

4.2.4. NewCommentViewController

ViewController encargado de añadir un nuevo comentario a un post. Al rellenar el campo de texto se realiza la petición a la API para crear un nuevo cometario en el post.

func sendCommentToBackoffice()

Es el método que se llama al aceptar el envío de un nuevo comentario. Llama al método correspondiente del singleton NetworkManager. Al recibir la respuesta, vuelve a la pantalla anterior. En este caso DetailPostViewController

4.2.5. ChatViewController

ViewController encargado de mostrar la pantalla de Chat de la aplicación. Desde él se envían y reciben los mensajes de los dos usuarios.

@IBAction func sendMessage(sender: AnyObject)

Es el método que se llama al pulsar en el envío de un mensaje en la pantalla de chat. Llama al método correspondiente del singleton NetworkManager. A su vez, introduce el mensaje en la tabla de mensajes.

4.2.6. ProfileViewController

ViewController encargado de mostrar tanto el perfil del propio usuario como el de los demá. Es el encargado de mostrar una lista de Diseases y de Treatments del usuario además de su información básica.

func isOwnProfile() -> Bool

Este método es llamado en varias ocasiones en la misma clase. Es el que determina si el usuario que se está viendo es el mismo que esta usando la aplicación o no. De esta manera se dejan de ver o se muestran componentes distintos.

@IBAction func segControlChanged(sender: AnyObject)

Callback que se llama cuando el componente UISegmentedControl de la pantalla ha cambiado de posición. En nuestro caso para cambiar entre las tablas de Diseases y de Treatments.

func goToChat()

Método que sólo se llama cuando el usuario que se está viendo no es el mismo que el que usa la aplicación. Este método lleva a la pantalla de chat ChatViewController para que el usuario pueda enviar mensajes al usuario del perfil que estaba viendo.

4.2.7. NetworkManager

Singleton encargado de mantener la coherencia de los datos de la aplicaci
Esta clase esta compuesta por muchos métodos parecidos para diferentes peticiones. En lugar de introducirlos todos vamos a poner un método genérico usando el término Model que los englobe todos.

func getModels(input: NSDictionary, completion: (result: NSArray) -> Void)

Método que se usa para llamar a la API y obtener todos los objetos del tipo Model (Post, Comment, Message, etc). Este mismo método llama a otro método descrito más abajo para serializar el modelo para posteriormente devolver un array de objetos Model.

func postNewModel(input: NSDictionary, completion: (result: JSON) -> Void)

Método que se usa para llamar a la API y hacer un POST de un objeto Model. Este método devuelve un JSON de respuesta del servidor para saber si ha ido bien o mal la petición.

func insertModelToCoreData(model: NSDictionary)

Método usado para insertar el objeto de tipo Model en la base de datos interna de la aplicación (CoreData), para que posteriormente si se quiere obtener los datos de algún modelo no se deba realizar una petición cada vez.

func serializeModel(models: JSON) -> NSMutableArray

Método que convierte los datos recibidos de la API a un array de objetos del modelo que se serializa. Así pues, convierte los datos JSON a objetos Model.

func getObjectWithKeyEqualToValue(table: String, key: String, value : AnyObject) -> NSManagedObject?

Método que devuelve un objeto del tipo NSManagedObject (CoreData) del tipo table concordante con la clave y el valor que se le inserta en el método.

4.3. Documentación API

A continuación vamos a exponer la documentación de la API. En ella podremos ver qué necesita cada petición cómo entrada y un ejemplo de un JSON de salida. La documentación la podremos leer de la forma:

Solicitud	Método	Parámetros entrada	JSON salida	Códigos de error	Descripción

4.3.1. Llamada GET para obtener todos los posts

/api/posts	GET	-	{
			"posts": [
			{
			"id": 1,
			"title": "My experience with Diabetes",
			"descriptionText": "It's been 15 years since the doctor...",
			"type": "Experience",
			"date": 1499263560000,
			"user_id": 1
			}
			]
			}
				-	Obtiene todos los posts

4.3.2. Llamada GET para obtener todos los comentarios de un post

/api/post/{id}	GET	id: Identificador del post	{
			"comments": [
			{
			"id": 1,
			"parent_post_id": 1,
			"descriptionText": "This is great! Right now I'm living...",
			"date": 1499263560000,
			"user_id": 1
			}
			]
			}
				404 Not found. Invalid id	Obtiene todos los comentarios de un post

4.3.3. Llamada GET para obtener la información de un usuario

/api/user/{id}	GET	id: Identificador del usuario	{
			"user": {
			"id": 1,
			"name": "Casey Neistat",
			"email": "casey@gmail.com",
			"dateBirth": 1499263560000,
			"bio": 1
			}
			}
				404 Not found.
				Invalid id
					Obtiene la información de un usuario

4.3.4. Llamada GET para obtener la lista de posts favoritos del usuario

/api/user/{id}/favs	GET	id: Identificador del usuario	{
			"posts": [
			{
			"id": 1,
			"title": "My experience with Diabetes",
			"descriptionText": "It's been 15 years since the doctor...",
			"type": "Experience",
			"date": 1499263560000,
			"user_id": 1
			}
			]
			}
				404 Not found.
				Invalid id
					Obtiene la lista de posts favoritos del usuario

4.3.5. Llamada GET para obtener la lista de chats de un usuario

/api/user/{id}/chats	GET	id: Identificador del usuario	{
			"chats": [
			{
			"id": 1,
			"user_to": 1,
			"user_from": 2
			}
			]
			}
				404 Not found.
				Invalid id
					Obtiene la lista de chats de un usuario

4.3.6. Llamada GET para obtener la lista de mensajes de un chat

/api/chats/{id}	GET	id: Identificador del chat	{
			"messages": [
			{
			"id": 1,
			"user_to": 1,
			"user_from": 2,
			"date": 1499263560000,
			"descriptionText": "Hey John!"
			}
			]
			}
				404 Not found.
				Invalid id
					Obtiene la lista de mensajes de un chat

4.3.7. Llamada POST para enviar un post

/api/post	POST	{
		"title": "My experience with Diabetes",
		"descriptionText": "It's been 15 years since the doctor...",
		"type": "Experience",
		"date": 1499263560000,
		"user_id": 1
		}
			{
			"success": 1
			}
				400 Bad Request.
				Wrong data.
					Envia un post

4.3.8. Llamada POST para enviar un comentario a un post

/api/post/{id}	POST	id: Identificador del post
		{
		"parent_post_id": 1,
		"descriptionText": "This is great! Right now I'm living...",
		"date": 1499263560000,
		"user_id": 1
		}
			{
			"success": 1
			}
				404 Not found.
				Invalid id.
				400 Bad Request.
				Wrong data.
					Envia un comentario a un post

4.3.9. Llamada POST para vincular un post favorito a un usuario

/api/user/{id}/favs	POST	id: Identificador del usuario
		{
		"post_id": 1
		}
			{
			"success": 1
			}
				404 Not found.
				Invalid id.
				400 Bad Request.
				Wrong data.
					Vincula un post favorito para un usuario

4.3.10. Llamada POST para enviar un mensaje a un chat

/api/chats/{id}	POST	id: Identificador del chat
		{
		"user_to": 1,
		"user_from": 2,
		"date": 1499263560000,
		"descriptionText": "Hey John!"
		}
			{
			"success": 1
			}
				404 Not found.
				Invalid id.
				400 Bad Request.
				Wrong data.
					Envia un mensaje a un chat

4.3.11. Llamada DELETE para borrar un post

/api/post/{id}	DELETE	id: identificador del post	{
			"success": 1
			}
				404 Not found. Invalid id.
				400 Bad Request. Wrong data.
					Borra el post con identificador id

4.3.12. Llamada DELETE para desvincular un post de un usuario

/api/user/{id}/favs	DELETE	id: Identificador del usuario
		{
		"post_id": 1
		}
			{
			"success": 1
			}
				404 Not found.
				Invalid id.
				400 Bad Request.
				Wrong data.
					Desvincula un post favorito para un usuario

4.4. Diseño final de la interficie

A continuación se muestran los diseños finales del proyecto. Estos se han creado siguiendo los iOS Human Interface Guidelines[8].

4.4.1. Pantalla principal

Esta es la primera pantalla de la aplicación. Desde ella se podrá acceder a la pantalla del inicio de sesión. Esta pantalla se mostrará al inicio de la aplicación.

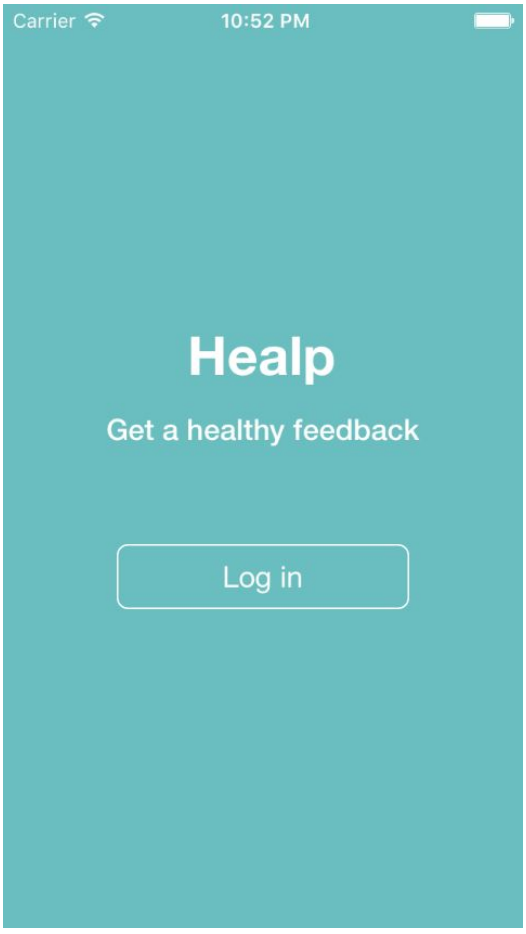


Imagen Pantalla principal

4.4.2. Pantalla listado de posts

En esta pantalla se muestra el listado de posts. En ella vemos una una lista de posts así como parte de la información. Mediante el botón "+" superior, podremos acceder a la pantalla de creación de un post. Usando el símbolo de filtro accederemos al menú para filtrar los posts. Mediante la barra de búsqueda, podremos buscar la información que queramos en todos los posts.

Imagen Pantalla listado de posts

4.4.3. Pantalla Filtrado de posts

En esta pantalla podremos filtrar los posts por categoría y tipo. De esa manera, se mostrarán los posts que desee el usuario. A esta pantalla se accede desde la pantalla del listado pulsando el botón de filtro o deslizando el dedo desde el borde izquierdo de la pantalla hasta el centro.

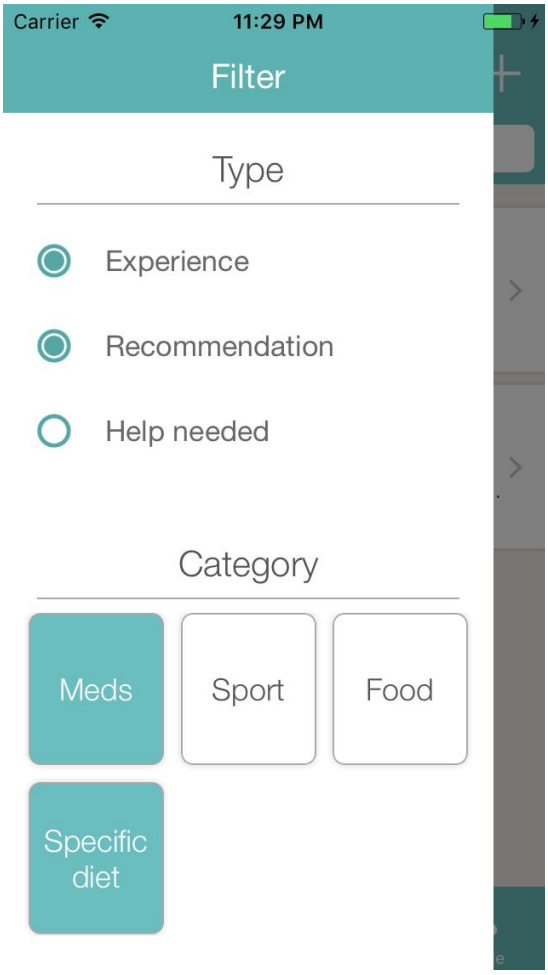


Imagen Pantalla Filtro de posts

4.4.4. Pantalla Detalle de un post

En la siguiente pantalla podremos ver la información completa de un post así como sus comentarios. Desde la flecha superior podremos realizar un comentario al post.

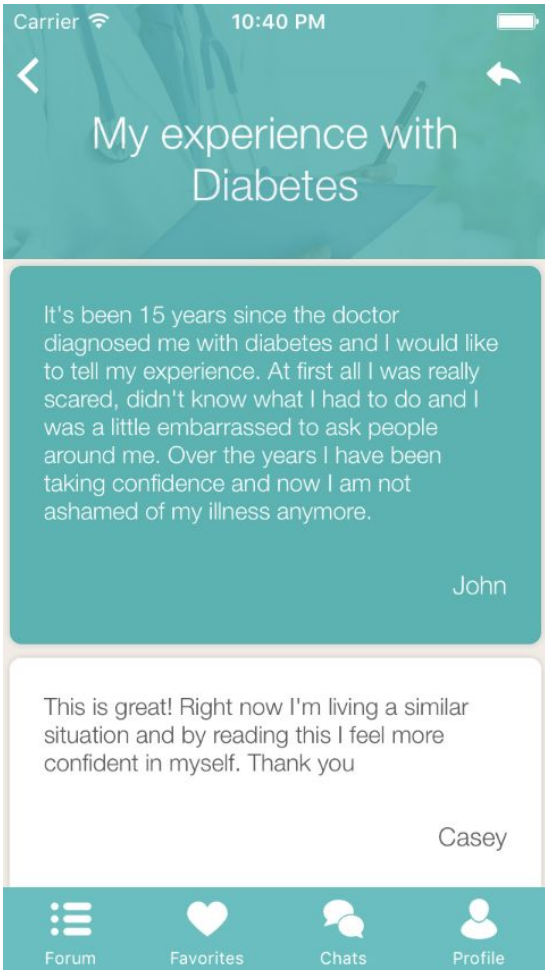


Imagen Pantalla detalle post

4.4.5. Pantalla Creación de un post

Desde esta pantalla podremos crear un post. Al rellenar la información y pulsando el botón, podremos enviar un post para verlo posteriormente en la lista.

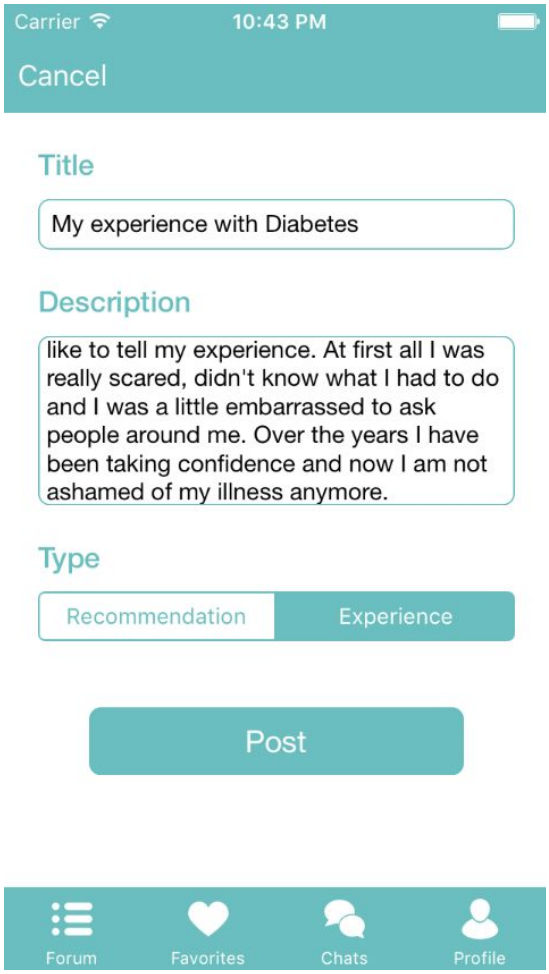


Imagen Pantalla añadir post

4.4.6. Pantalla Responder a un post

Mediante esta pantalla podremos responder a un post. Deberemos rellenar el campo con el texto que queremos que sea la respuesta.

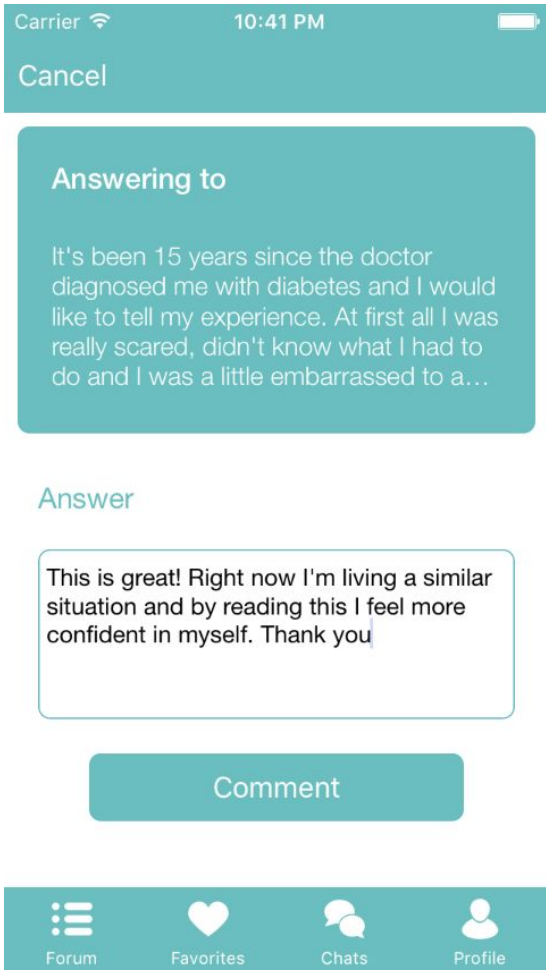


Imagen Pantalla responder post

4.4.7. Pantalla Posts favoritos

En esta pantalla se nos mostrarán los posts que hemos marcado como favoritos. Desde esta pantalla tambien podremos acceder al detalle de un post.

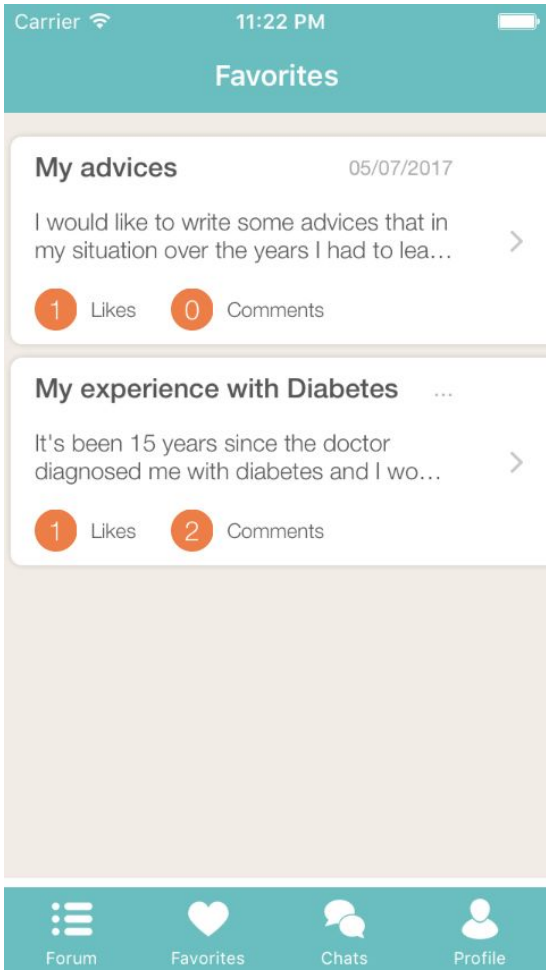


Imagen Pantalla favoritos

4.4.8. Pantalla Perfil

En la pantalla podremos ver la información de perfil del usuario. En la parte inferior nos mostrará la lista de dolencias o de tratamientos que esta padeciendo o siguiendo el usuario. Mediante los botones “Add new disease” o “Add new treatment” pasaremos a las respectivas pantallas para añadir una dolencia o un tratamiento.

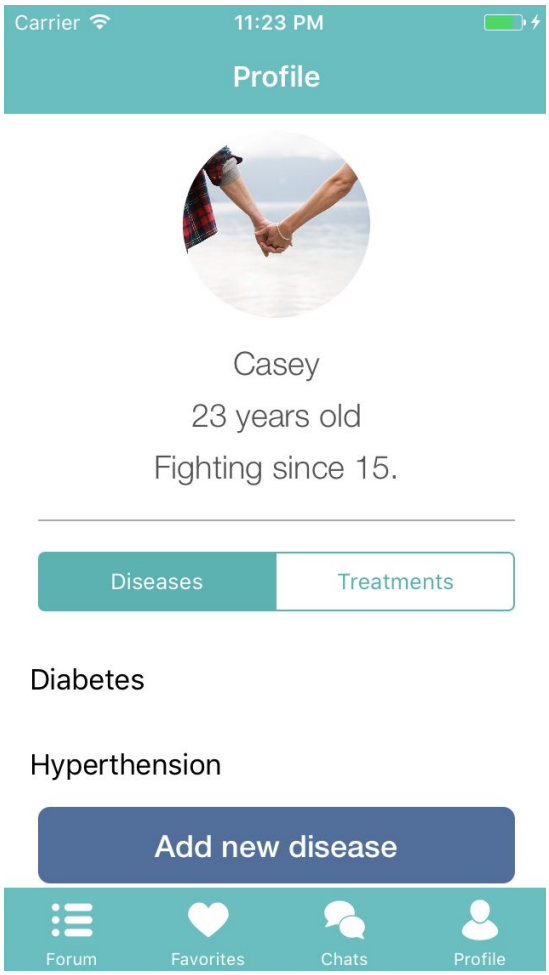


Imagen Pantalla perfil

4.4.9. Pantalla Añadir Disease

Esta pantalla sirve para añadir una dolencia al usuario. El usuario introduce una dolencia que tiene que se mostrará posteriormente en su perfil.

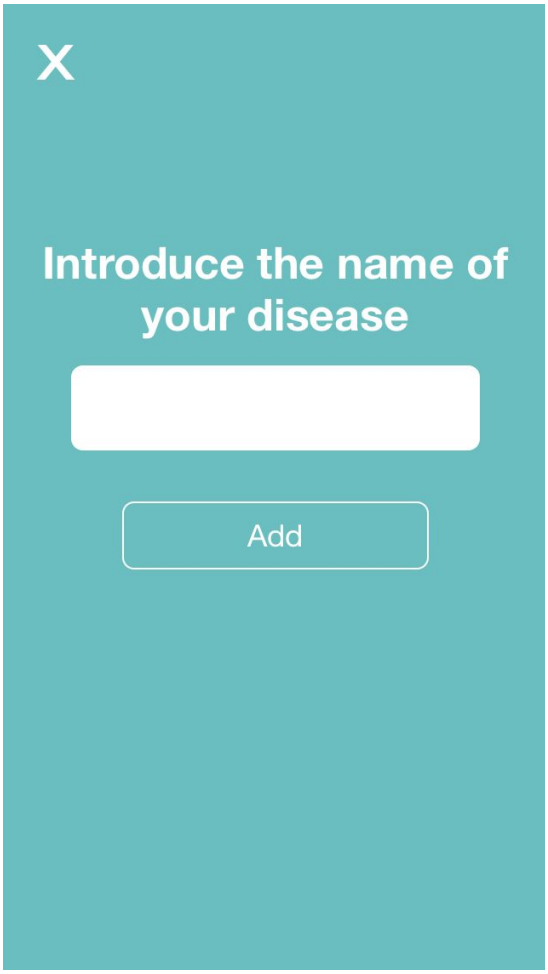


Imagen Pantalla Añadir Disease

4.4.10. Pantalla Añadir Treatment

Pantalla que permite al usuario introducir un tratamiento que está siguiendo. Este tratamiento debe tener asociada una categoría, la cual se mostrará en el desplegable que aparecerá al introducir la categoría.

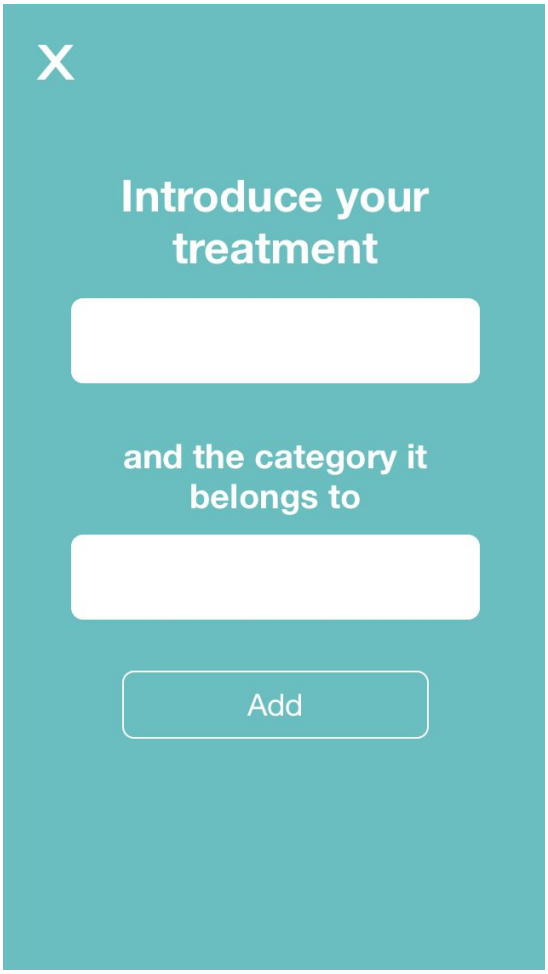


Imagen Pantalla Añadir Treatment

4.4.11. Pantalla Lista de chats

En la siguiente pantalla obtendremos una lista de los chats con los que hemos iniciado una conversación. Al hacer click en cualquiera de ellos nos llevará a la pantalla de chat.

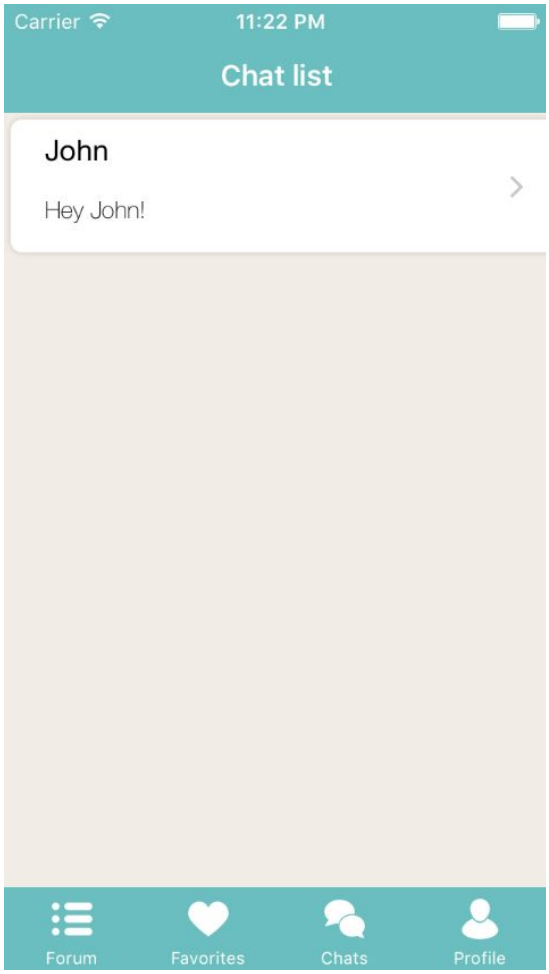


Imagen Pantalla chats

4.4.12. Pantalla Chat

Esta pantalla nos muestra el historial de mensajes que hemos tenido con la persona. Desde ella recibiremos los mensajes y podremos enviarlos.

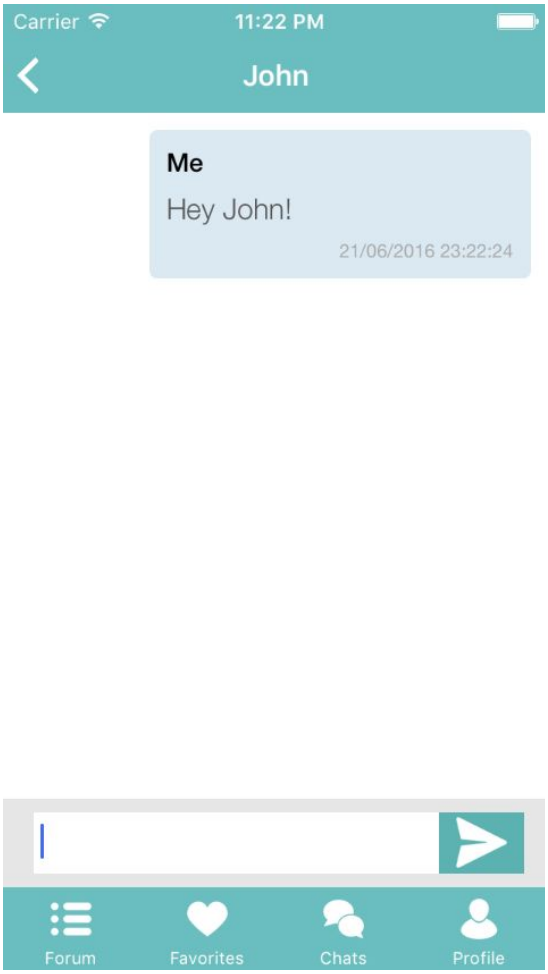


Imagen Pantalla mensajes

5. Tecnologias usadas

5.1. Tecnologías usadas para el desarrollo de la aplicación

Para el desarrollo de la aplicación se han usado las siguientes tecnologías:

Xcode 8.0
Es el entorno de desarrollo integrado que nos proporciona Apple para poder desarrollar las aplicaciones para iOS. Tambien se ha usado el *Interface Builder* contenido en Xcode para realizar el desarrollo visual de las pantallas.

Swift 3
Se ha usado el lenguaje de programación Swift. La razón de usar este lenguaje y no Objective-C ha sido por motivación personal y ganas de aprender el lenguaje. Este proyecto ha sido la excusa para poder aprender en gran medida este nuevo lenguaje sacado por Apple hace 2 años.

Autolayout
Se ha usado la tecnología Autolayout para el diseño de las pantallas. Esta tecnología, a diferencia del Autoresizing consiste en aplicar relaciones entre los componentes de la pantalla. Se puede determinar el comportamiento de un componente de la pantalla mediante un *tipo de relación*, una *constante*, un *multiplier* y una *prioridad*.

Cocoapods
Es un gestor de dependencias de Swift y Objective-C desarrollada en Ruby. Se ha usado esta herramienta para gestionar las dependencias del proyecto: añadir las librerias que vamos a mencionar a continuación.

5.1.1. Librerías

Alamofire 4.0 [10]
Librería desarrollada en Swift que nos permite realizar peticiones HTTP cómodamente. Mediante esta librería, realizamos las peticiones a la API. Entre sus características, destacamos la facilidad para realizar peticiones en que el parámetro es un JSON.

MGSwipeTableCell [11]
Librería desarrollada en Objective-C que subclasifica la clase *UITableViewCell* de Apple y nos permite añadirle acciones y transiciones a las celdas. Mediante ella hemos conseguido que realizando el gesto *Swipe* de derecha a izquierda en un celda, nos permita borrar y darle a like a un post.

SwiftyJSON [12]
Librería desarrollada en Swift que nos permite trabajar de manera mucho más fácil con JSONs. En el caso de nuestro proyecto, hemos combinado de manera fácil la librería Alamofire y SwiftyJSON para realizar las peticiones a la API. De esta manera, nos simplifica en gran medida los parámetros de entrada y de salida.

LGRefreshView 1.0.0 [13]
Librería desarrollada en Objective-C que nos permite añadir la funcionalidad *Pull-To-Refresh* a una tabla. De tal manera, deslizando la tabla hacía abajo, realizamos una petición de GET a la API para refrescar los posts de la tabla.

SVProgressHUD [14]
Librería desarrollada en Objective-C que nos permite mostrar un HUD de descarga de una manera muy sencilla. Lo usamos cuando realizamos peticiones a la API y queremos informar de al usuario de que se está realizand una tarea.

OpinionzAlertView [15]
Librería desarrollada en Objective-C que nos permite mostrar dialogos de confirmación. Lo usamos cuando debemos mostrar un dialogo de confirmación, ya sea al crear un post o al borrarlo por ejemplo.

5.2. Tecnologías usadas para el desarrollo de la API

Para el desarrollo de la API se han usado las siguientes tecnologías:

PhpStorm 10
Es el entorno de desarrollo integrado de la compañia Jetbrains para Php. Con el es se puede desarrollar un proyecto Laravel de forma muy cómoda. Con el *Inspector de Archivos* y el autocompletado para Php te ayudan a desarrollar de una forma muy rápida un proyecto de este tipo.

Laravel 5.2 [16]
Hemos usado el framework PHP Laravel para realizar la API. Se ha escogido este framework debido al anterior uso del mismo en otros proyectos. Con él, se puede crear una API de una forma bastante más simple que realizarla con otras tecnologías. Además, Laravel usa un sistema de *Routing* muy útil para asignar las rutas de las peticiones a funciones en los controladores del proyecto. Otra cosa muy útil de Laravel es que usa *Eloquent ORM*, que es una implementación para que sea mucho más fácil tratar con el modelo y la base de datos.

Imagen Logo Laravel

PostgreSQL
Es el sistema de gestión de base de datos relacional que se ha escogido, ya que se integra mucho mejor con Laravel que otras sistemas, como MySQL. Con la ayuda de *Eloquent ORM* y las *Migrations* de Laravel no ha sido necesario realizar una gestión de bases de datos propiamente dicha, ya que Laravel se ha encargado de eso por nosotros.

5.3. Tecnologías usadas para el control de versiones

A lo largo del desarrollo se ha usado GIT como sistema de control de versiones. Se ha creado un repositorio en GitHub tanto para la aplicación como para el proyecto de la API. En ambos proyectos se ha usado el sistema Gitflow.

Imagen Gitflow [17]

Como podemos observar en la imagen, el desarrollo se realiza en la rama *Develop*. Para cada funcionalidad, deberemos crear una rama *Feature* en la que desarrollaremos una parte o toda la funcionalidad del proyecto y posteriormente, cuando hayamos acabado con la funcionalidad, la integraremos en la rama *Develop*.

En el momento que queremos generar una versión de la aplicación, deberemos generar una *Release*. En esta rama tendremos una versión que, en caso de ser una empresa y/o tener clientes, podriamos entregar. Cuando cerramos la *Release*, automáticamente el sistema nos integra esa rama con *Develop* y con *Master,* y así nos aseguramos que en la rama *Develop* tenemos los últimos cambios y en la rama *Master* la versión.

Posteriormente a realizar la versión, se podrá seguir desarrollando funcionalidad en la rama *Develop* con el sistema anteriormente mencionado. Sin embargo, si ocurriera algún fallo que debiera arreglarse en la versión generada, deberíamos recurrir a la rama *Hotfix* que nos permitirá arreglar fallos de esa versión, y al acabarlo, integrarla con *Master* y con *Develop* para que esté actualizado con los cambios.

La rama *Hotfix* nos sirve básicamente para no tener que deshacer los cambios que llevamos desarrollados en *Develop*. Ya que si por ejemplo, hemos hecho una release de una versión 1.0 y ya estamos desarrollando la versión 2.0 pero aparecen fallos de la 1.0, deberemos hacer un *Hotfix* para la version 1.0 sin tener que deshacer las funcionalidades que llevamos desarrolladas de la version 2.0.

6. Planificación y sostenibilidad

6.1. Planificación

En este apartado se realizará una explicación de cual ha sido la planificación y la división de tareas para cada *Sprint* para el desarrollo del proyecto.

El proyecto inicialmente se estimó que duraría desde el mes de Febrero de 2016 y Julio de 2016, en el cual Febrero y Marzo se iban a dedicar a la asignatura de GEP (Gestión de Proyectos). Tambien se planificó que el desarrollo del proyecto se dividiría en 6 sprints.

La duración de cada Sprint es de dos semanas. Cada uno por lo general está compuesto por un *Sprint planning*, una fase de *desarrollo*, una fase de *testing* para las funcionalidades desarrolladas, una fase de *arreglo de errores* y una última fase de *integración.* Sin embargo, no todos tienen estas fases ya que no todos los *Sprints* han sido de desarrollo, ya que por ejemplo el primer Sprint y el último han sido de investigación y documentación respectivamente.

6.1.1. Sprint 1

Este sprint principalmente ha sido dedicado a la investigación. La asignatura de GEP ha sido muy útil para la parte de investigación ya se disponía de mucha información de la asignatura de GEP. No solo ha servido para la investigación del tema del proyecto sino tambien para la investigación de herramientas y tecnologías a usar en el desarrollo del proyecto. Ha sido en este Sprint en el que se ha decidido por ejemplo usar Laravel como framework de backend y iOS como sistema operativo de la aplicación.

Además de lo descrito anteriormente se han realizado *wireframes* básicos de la aplicación para tener una idea del producto final que se quería. Por último tambien se ha realizado un backlog con los requisitos y funcionalidades que se querían para el producto y se ha ordenado por prioridad.

6.1.2. Sprint 2

Este sprint se ha dedicado al desarrollo de las primeras funcionalidades, todas relacionadas con el objeto principal del proyecto: los posts. Así pues, por parte de la aplicación se ha desarrollado la lógica que tienen los posts así como las pantallas. Las funcionalidades principales han sido el mostrar la lista de posts, añadir un post a la aplicación y realizar un comentario.

Por parte de la API también se ha desarrollado esa parte del proyecto. Se ha realizado el modelo de los posts y comentarios y se ha desarrollado la lógica que siguen esas dos entidades en la parte de *backend.* Sin embargo, no se ha integrado la aplicación con la API

6.1.3. Sprint 3

Este sprint se ha dedicado al desarrollo de la parte de comunicación de la aplicación. Así pues se ha desarrollado la lógica que siguen los chats y los mensajes y tambien la información de los usuarios.

Al igual que en el sprint anterior se ha desarrollado la misma parte del proyecto de la aplicación y la lógica que hay entre las entidades. Además también se han integrado con las dos entidades que ya había creadas del *sprint 2.* En este sprint se ha seguido sin integrar la API con la aplicación.

6.1.4. Sprint 4

Este sprint es el último que se ha dedicado integramente al desarrollo. En este sprint se han acabado las funcionalidades de la aplicación que faltaban como podrían ser la búsqueda de posts y la lógica de favoritos y se ha integrado con la aplicación.

En la parte de la API tambien se han desarrollado las mismas funcionalidades que en la aplicación y se ha realizado la integración con la aplicación. De esa manera, en este sprint hemos tenido ligada la aplicación con la API lo que significa que todas los dispositivos con la aplicación compartían los mismos datos.

6.1.5. Sprint 5

Este sprint se ha dedicado a la parte visual del proyecto. He realizado el rol de diseñador en este sprint y se ha realizado el diseño de la aplicación. Además, se han realizado ciclos completos con la aplicaciones simulando un uso real de la aplicación para encontrar posibles fallos. Tambien se han solucionado las incidencias que se han encontrado.

6.1.6. Sprint 6

Este sprint es el sprint que actualmente se está llevando a cabo. Es el sprint donde se realiza la documentación del proyecto. Esta parte del proyecto consiste en desarrollar la memoria del proyecto así como preparar la defensa del proyecto en el tribunal.

6.2. Gestión económica

A continuación se detallan los costes del proyecto

6.2.1. Costes recursos humanos

Debido a que el proyecto es realizado solamente por una persona, la remuneración asociada a cada rol coresponde con el precio de hora mínimo que establece el informe de *Page Personnel*[18] del año 2016.

Rol	Horas	Remuneración (€/h)	Coste (€)
Responsable Proyecto	201	17,5	3.517,5
Analista	30	12	360
Programador	270	9	2.430
Tester	70	11	660
Total	561		**6967,5**

Tabla costes recursos humanos

6.2.2. Costes de hardware

Se estima que los recursos de hardware que se han usado tienen una vida útil de 4 años y estipulamos que un año tiene 250 días laborables. De esa manera, establecemos el coste de amortización los 250 días a un ritmo de trabajo de 8 horas diarias.

Coste amortización = Precio / (4 años * (250 días * 8 horas/día))

Hardware	Precio (€)	Horas	Coste amortización (€/h)	Coste (€)
Macbook Pro 15” con OSX El Capitán	2.249,00	471	0,281125	133
iPhone 5S 16 GB	650	110	0,08125	9
Total				**142**

Tabla costes hardware

6.2.3. Costes generales

Ya que la empresa ya tiene contratada fibra óptica y el coste no depende del uso que se le haga a esta, no vamos a contar con el coste de la conexión a internet. Sin embargo, sí que vamos a contar la electricidad que gasta la carga de batería del ordenador. No vamos a contar la carga de batería del smartphone ya que al cálculo, son unos pocos céntimos de euro.

El coste que tampoco vamos a contar es el del transporte hacia la empresa, ya que el proyecto no tiene influencia en la adquisición del abono de tren (dado el caso), pues es independiente al proyecto.

Descripción	Precio	Cantidad	Coste estimado (€)
Consumo energético	0,15 €/kWh	Consumo diario 0,1 kWh	10
Impresiones en papel	100	0,05 €/impresion	5
Software	0	-	0
Desplazamiento a Sfy	0	-	0
Conexión a internet, local, agua, ...	0	-	0
Total			**15**

Tabla gastos de recursos

6.2.4. Coste total

Concepto	Coste (€)
Recursos humanos	6.967,5
Hardware	142
Costes generales	15
Coste total	**7124,95**

Tabla coste total

6.3. Sostenibilidad y compromiso social

6.3.1. Económica

El proyecto en si, podría haberse realizado en menos tiempo contando con más recursos humanos, pero al ser el responsable del proyecto el que ha ejercido todos los roles, el tiempo que ha tardado el proyecto en realizarse es superior.

De momento, no se prevé ninguna colaboración con otros proyectos, pero no se descarta que empresas externas de ámbito médico puedan tener un interés en el producto final.

6.3.2. Social

Desde hace tiempo, hay una necesidad real en el mundo de mejorar la situación de los pacientes. El producto final intenta poner un granito de arena en mejorar la calidad de vida de las personas que padecen enfermedades, sobretodo crónicas, y a su vez, ahorrar a las organizaciones relacionadas con la salud mucho dinero que se gasta por culpa de pacientes que no siguen el tratamiento que deberían.

6.3.3. Ambiental

El impacto que cabe destacar sobre el ambiente y el producto final es todo el ahorro de recursos por parte de organizaciones relacionadas con la salud que harían. Que los pacientes tengan una mejor calidad de vida se traduce en mucho menos gasto por parte de hospitales en recursos tales cómo instrumental médico.

8. Metodologia y rigor

Para el desarrollo del proyecto se ha usado la metodología de desarrollo ágil. Como hemos visto en la planificación, hemos dividido el proyecto en sprints de dos semanas y cada sprint tenía un sprint planning para planificar qué se iba a hacer en ese sprint. Los objetivos de cada sprint se marcaban en base a las funcionalidades sacadas del backlog y sus prioridades.

Para validar que cada sprint estaba desarrollandose de la forma planificada, cada finalización de sprint estaba compuesta por un *Sprint review*, que consistía en una reunión con el tutor del proyecto.

A su vez, al final del desarrollo del producto se han realizado tests de ciclo completo para validar que el producto final era usable e intuitivo. Estos tests, explicado de manera simple, consisten en realizar los ciclos completos que harían los usuarios finales de tal manera que si en algún momento del ciclo existe alguna incongruencia, se pueda cambiar/solucionar.

Adicionalmente, para la validación de código, se han realizado tests unitarios de funcionalidades específicas que nos han validado que el comportamiento de partes del código es el esperado. Además también nos ha obligado a realizar una mayor modularización del código, lo que conllevará un mejor entendimiento de este en un futuro.

9. Conclusión

Todo proyecto empieza con un reto, un reto en el cual quieres aportar tu visión y tu tiempo. La idea de este proyecto empezó con unas pocas palabras: *patient adherence*. Al escuchar eso por primera vez de la boca de una compañera de la empresa, y como no sabía qué era, le pedí que me lo explicara. Después de que me lo explicara, en mi interior resumí en una frase el problema que ocurría: *los pacientes no quieren seguir su tratamiento*.

La investigación sobre este tema duró muchos días, ya que aunque no uno no se lo imagine, este problema ha perdurado durante muchísimos años. La pregunta que se formula uno para este reto podría parecerse a: *¿Cómo puedo hacer que los pacientes quieran seguir su tratamiento?*. Al estudiar las razones de que no quieran seguir su tratamiento me di cuenta de que no es fácil responder a esta pregunta. Las razones de cada persona son diferentes unas de otras. Algunas personas no quieren seguir porque no quieren estar ligadas a un tratamiento, otras no quieren porque los efectos secundarios de su medicación le sientan peor que la propia enfermedad y otras simplemente se olvidan de que tienen un tratamiento que seguir.

Después de estudiar las razones y ver que eran tan variadas, decidí centrarme en dos principales: la motivación del paciente y el soporte que estos obtienen. Con compañeros de la empresa pensamos durante un tiempo cómo mejorar el las ganas que de un paciente a seguir un tratamiento y llegamos a la conclusión de que la mejor manera es mediante gente que sepa exactamente por lo que está pasando. De esa manera salió la idea de intentar hacer una comunidad de gente donde intenten motivarse entre ellos.

Personalmente creo ha sido un tema muy difícil sobre el que hacer una herramienta de software. Creo que los objetivos funcionales de la aplicaciones han sido cumplidos ya que la aplicación hace lo que se propuso al inicio. Sin embargo, los objetivos del proyecto, como puede ser mejorar la calidad de vida de los pacientes, crear una comunidad donde los pacientes puedan expresar sus dudas y experiencias, no depende del proyecto en si, sino de las personas que van a usarlo.

10. Referencias

[1] Patient Medication adherence: Measures in daily practice. Beena Jimmy, Jimmy Jose. Aceptado 9 Abril 2011. Oman Medical Journal 2011 Vol. 26.

[2] Adherence to Long-Therm Therapies: Evidence for Action, World Health Organization 2003.

[3] Adheretech Wireless Pill Bottles [en línea]. [Consultada 29 Febrero 2016] <https://adheretech.com>

[4] Medisafe - Never forget your pills again [en línea]. [Consultada 29 Febrero 2016] <http://www.medisafe.com>

[5] MedHelper - Mobile Prescription App [en línea]. [Consultada 29 Febrero 2016]<http://medhelperapp.com>

[6] e-pill Medication Reminders [en línea]. [Consultada 29 Febrero 2016]<http://www.epill.com>

[7] Dosecast - Montuno Software [en línea]. [Consultada 29 Febrero 2016]<http://www.montunosoftware.com/products/dosecast/about>

[8] iOS Human Interface Guidelines [en línea]. [Consultada 24 Junio 2016] <https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/>

[9] Model-View-Controller. iOS Developer Library [en línea]. [Consultada 24 Junio 2016] <https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html>

[10] Alamofire. Github. [en línea]. [Consultada 24 Junio 2016] <https://github.com/Alamofire/Alamofire>

[11] MGSwipeTableCell. Github. [en línea]. [Consultada 24 Junio 2016] <https://github.com/MortimerGoro/MGSwipeTableCell>

[12] SwiftyJSON. Github. [en línea]. [Consultada 24 Junio 2016] <https://github.com/SwiftyJSON/SwiftyJSON>

[13] LGRefreshView. Github. [en línea]. [Consultada 24 Junio 2016] <https://github.com/Friend-LGA/LGRefreshView>

[14] SVProgressHUD. Github. [en línea]. [Consultada 24 Junio 2016] <https://github.com/SVProgressHUD/SVProgressHUD>

[15] OpinionzAlertView. Github. [en línea]. [Consultada 24 Junio 2016] <https://github.com/Opinionz/OpinionzAlertView>

[16] Laravel Homepage [en línea]. [Consultada 24 Junio 2016] <https://laravel.com>

[17] GIT Flow. Derek's SQL Server Blog. [en línea]. [Consultada 24 Junio 2016] <*http://www.derekevans.us/2014/11/git-flow/*>

[18] Estudios de remuneración 2016. Tecnología [en línea]. [Consultada 24 Junio 2016]
<http://www.pagepersonnel.es/sites/pagepersonnel.es/files/er_tecnologia16.pdf>